IERG4210 Web Programming and Security (2015 Spring)
Assignment Marking Guidelines

Revision History

v6.0	May 1	Released Phase 7B Requirements
v5.0	Mar 28	Released Phase 5, 6, 7A Requirements
v4.1	Mar 14	Released Phase 4B Requirements
v4.0	Mar 1	Released Phase 4A Requirements
v3.0	Feb 5	Released Phase 3 Requirements
v2.0	Jan 28	Released Phase 2 Requirements
v1.0	Jan 16	Created this document; Released Phase 1 Requirements

General Guidelines

The assignment is designed to let students practice what they have learned in the course. Students must be aware of web application security throughout the web development. The whole assignment is split into 7 phases, leading all the way to a fast, secure, and user-friendly shopping website upon completion. Students can take a real-world website, walmart.com, as a reference. In the assignment, students are expected to understand and apply proper security design principles and programming skills, regardless of what libraries (e.g., jQuery) the students would like to use. The marking checklist in the next page is written in a minimal-viable and result-oriented basis, and thus students can unleash their creativity in building more features. For detailed guidance, students should refer to both tutorial and lecture notes.

Submission Policy

Each student is required to maintain their source code and any resources (e.g., images, css and js files) in an assigned private repository **shop[01-80]** at github.com/ierg4210. While latest changes are always maintained in the *master* branch, students are required to branch out a snapshot titled as **phase[1-7]** from its *master* for each phase. Hence, TAs will pull from the particular branch and only take that into account for inspection. Technical details can be found in Tutorial 2.

Each phase is associated with a firm submission deadline.

- *Early Submission Incentive* – For every 48-hour advanced submission in one phase, the deadline for phase 5 or 6 can be extended by 24-hour, and no part thereof is accepted. For instance, submitting 100 hours before the phase 1 deadline will gain an extension of 48 hours for either phase 5 or 6 deadline of your choice.
- *Late Submission Penalty* – Late submission will incur deduction of $(10\%)^{1/n}$ from your scored points, where n is the round-up number of days delayed (e.g. 9 hrs late → 10%, 25 hrs late → ~31.6%, 49 hrs late → ~46.5%, and so forth).
- *Interim Demonstration* – Students' submissions will be randomly sampled by TAs for inspection. If a student is found unable to complete 70% of the requirements in any single phase from 1 to 5, s/he is required to see TA, and will be arranged to give an interim demonstration (time and venue TBD). If the student fails to complete 70% of the requirements by the time his/her interim demonstration is given, the case will be escalated to the department for resolution. Students capable of meeting deadlines and 70% of the requirements can safely ignore this policy.
- *Final Demonstration* – Students will sign up for a timeslot to demonstrate their websites to a marker, who will then grade it according to the checklist. The marker will further evaluate the student's understanding with two questions.

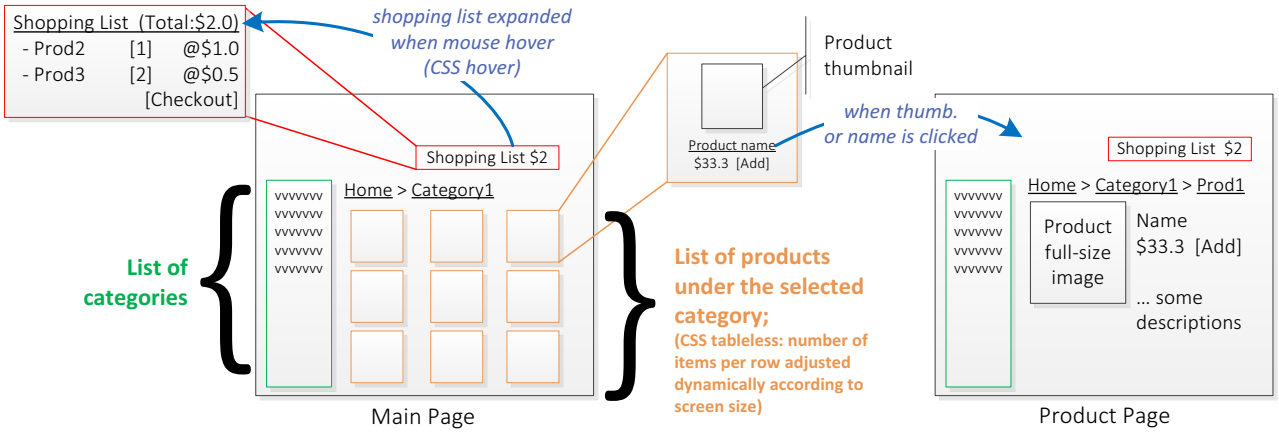
Honesty in Academic Work

CUHK places very high importance on honesty in academic work submitted by students, and adopts a policy of *zero tolerance* on cheating in examinations and plagiarism. Students are NOT allowed to submit anything that are plagiarised. Therefore, we treat every assignment our students submit as original except for source material explicitly acknowledged. We trust that students acknowledge and are aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/.

IERG4210 WEB PROGRAMMING AND SECURITY (2015 SPRING)
ASSIGNMENT MARKING CHECKLIST V5.0

PHASE 1: LOOK AND FEEL (DEADLINE: JAN 26, 2015, 5PM) (SUBTOTAL: 12%)

A designer has drafted a layout as follows, which outlines some fundamental features of a shopping website. In this phase, you will create a mock-up by hardcoding the website with **dummy categories and products**.



1. HTML: Make use of semantic HTML throughout the whole assign. _____ / 1%
   - <header>, <nav>, <footer>, <article>, <section>, <ul>,
2. CSS: Clean separation of HTML, CSS and JS code and files throughout the whole assign. _____ / 2%
   - No inline CSS and JS are allowed
   - No styling in HTML, e.g. , , etc
   - Tolerance: < 5 exceptions
3. *Main page* demonstrates the use of "CSS tableless" *product list* _____ / 2%
   - Each product has at least its own thumbnail, name, price and *addToCart* button
   - When the thumbnail or name is clicked, redirect to the corresponding product page
4. *Main page* demonstrates the use of "CSS hover" *shopping list* _____ / 2%
   - When displayed, it will cover any elements behind
   - Input boxes are used for inputting quantity of each selected product
   - A checkout button is used to submit the list to Paypal
   - The shopping list is displayed in both main and product pages
5. *Product page* provides product details _____ / 2%
   - To show a full-size or bigger image, name, description, price, and *addToCart* button
6. Both main and product pages should include a navigation menu _____ / 2%
   - e.g., Home or Home > Category1 or Home > Category1 > Product1
   - Those underscored are hyperlinks that redirect users to an upper hierarchy
7. Branch out **phase1** in your repository, where TAs can checkout for inspection _____ / 1%

PHASE 2: TESTING ENVIRONMENT SETUP (DEADLINE: FEB 4, 2015, 5PM) (SUBTOTAL: 11%)

In this phase, you are required to setup a local development environment and also a remote deployment environment. Some guidance will be given in tutorial 4.

1. Create a web application and server using Node.js
   - Initialize a new node project, and install the following npm packages _____ / 1%
     - as dependency: `express` and `express-handlebars`
     - as development dependency: `supervisor`
   - Hosting your server locally _____ / 1%
     - Use `supervisor` to run your app locally to avoid manual restarting upon code changes
     - Server accessible at http://localhost[:3000] or any port number

- Serve static files using the express.static middleware _____ / 2%
  - Static image files accessible through http://localhost[:3000]/**images/**
  - Static JavaScript files accessible through http://localhost[:3000]/**lib/**
  - Static CSS files accessible through http://localhost[:3000]/**css/**
- Serve dynamic pages with a *server-side* JS templating engine _____ / 3%
  - Based on the simple basic example of ExpressHandlebars (sample code)
    - Move the common UI code from your two HTML into a main template layout
    - Hence, core contents of your pages will be combined with the main template
  - The *main page* accessible through http://localhost[:3000]/
  - The *product page* accessible through http://localhost[:3000]/**product**

2. Deploy your code to Amazon Elastic Beanstalk** (EB) _____ / 4%
   - Create a EB environment for Node.js (application name: shopXX-ierg4210)
   - Deploy your application to the EB environment#
   - Remotely accessible through the given application URL
   - Branch out **phase2** in your repository, where TAs can checkout for inspection
     - `.gitignore` configured to exclude committing any installed node and virtualenv packages

** *This will incur charges:*
(1) Apply the free US$100 AWS credit code, to be distributed by TA
(2) After the first deployment, configure to use single instance to save cost during development
(3) Terminate any unused resources (during/after this course). You're responsible for charges beyond your free quota

PHASE 3A: BACKEND PORTAL AND MANAGEMENT (DEADLINE: FEB 18, 2015, 5PM) (SUBTOTAL: 8%)

In this phase, you will build a backend portal for administrators to manage your products.

1. Create a database with the following structures _____ / 1%
   - A table for *categories*
     - Required columns: *catid (primary key, unique), name*
     - Data: at least 2 categories of your choice
   - A table for *products*
     - Required columns: *pid (primary key, unique), catid, name, price, description*
     - Data: at least 2 products for each category
2. Create a backend management portal for admin
   - Design several HTML forms to add, change*, and delete a category in DB _____ / 2%
     - Input field for category's name
   - Design several HTML forms to add, change*, and delete a product in DB _____ / 5%
     - Dropdown menu to choose and change the category (using <select>)
     - Input field for product's name
     - Input field for product's price (supporting 2 decimal places)
     - Textarea for product's description
     - File field for product's photo
       (Format: jpg/gif/png, Size: <=10MB, Name: *pid*.[jpg|gif|png])

   * The original values must be restored back to the inputs before a change (display the photo nearby)
3. Branch out **phase3a** in your repository, where TAs can checkout for inspection

PHASE 3B: FRONTEND PRESENTATION (DEADLINE: FEB 27, 2015, 5PM) (SUBTOTAL: 16%)

In this phase, you will implement the shopping cart that allows users to shop around your products. This phase is designed to let you practise data presentation and AJAX programming.

1. Populate the *main page*'s contents from DB with Handlebars or ExpressHandlebars
   - Populate the *category list* from DB _____ / 1%
   - Based on the category picked by user, populate the corresponding *product list* from DB _____ / 3%
     - Reflect catid in the URL (i.e., /?catid=[x], or simply /[x])
     - The corresponding product list is shown upon accessing the new URL in a new tab
2. Populate the *product page*'s contents from DB with Handlebars or ExpressHandlebars _____ / 2%

- Display the details of a product according to its DB record

3. Using JavaScript, dynamically update[#] the *shopping list*
   - When the *addToCart* button of a product is clicked, add it to the shopping list _____ / 1%
     - Adding the same product twice adds up quantity, do not display two rows of record
   - Once a product is added,
     - Users are allowed to update its *quantity* and delete it with a number input, or two buttons for increment and decrement _____ / 1%
     - Store its *pid* and *quantity* in the browser's `localStorage` _____ / 2%
     - Get the *name* and *price* over AJAX (with *pid* as input) _____ / 3%
     - Calculate and display the total amount at the client-side _____ / 1%
   - Once the page is reloaded, the *shopping list* is restored _____ / 2%
     - Page reloads when users browse another category or visit the product detail page
     - Populate and retrieve the stored products from the `localStorage`

   [#] The whole process of *shopping list* management must be done without a page load
4. Branch out **phase3b** in your repository, where TAs can checkout for inspection
   - Include a README.md file in your repo and document your application URL

PHASE 4A: WEB APPLICATION SECURITY (DEADLINE: MAR 20, 2015, 5PM) (SUBTOTAL: 22%)

In this phase, you will protect your website against many popular web application security threats.

1. No XSS Injection and Parameter Tampering Vulnerabilities in the whole webapp
   - [UX Enhancement Only] Proper and vigorous client-side input restrictions for all forms _____ / 1%
   - Proper and vigorous server-side input validations sanitizations and for all forms _____ / 2%
   - Apply Content Security Policy _____ / 2%
     - Given the least privilege principle, apply the tightest policy (e.g., disable inline)
     - Hints: CSP Generator
   - Apply context-dependent output sanitizations _____ / 2%
     - Apply context-parser-handlebars or express-secure-handlebars (release before March 9)
     - If you concatenated user inputs without handlebars, apply xss-filters manually
2. No SQL Injection Vulnerabilities in the whole webapp _____ / 2%
   - Apply prepared SQL statements
3. No CSRF Vulnerabilities in the whole webapp _____ / 2%
   - Apply and validate secret nonces (must be user-specific and random) using csurf
   - Every form and state-changing requests must defend against Traditional and Login CSRF
4. No information leaking (e.g., thru global variables) outside scopes of Node.js request handlers _____ / 1%
5. Authentication for Admin Panel
   - Create a user table (or a separate DB with only one user table) _____ / 1%
     - Required columns: *userid (primary key), username, password, admin**
     - Data: at least 2 users of your choice
     - Security: Passwords must be properly salted and hashed before storage

     ** 0 indicates non-admin, 1 indicates admin*
   - Build an *login page* at `/admin/login` that asks for *email* and *password* _____ / 2%
     - Upon proper validations and authentication, redirect the user to the *admin panel*
     - Otherwise, prompt for errors (i.e. your email or password is incorrect)
   - Maintain an authentication token using express-session
     - Cookie name: `auth`; value: a signed session id; property: `httpOnly` _____ / 1%
     - Cookies persist after browser restart, and last for at most 3 days _____ / 1%
     - No Session Fixation Vulnerabilities (rotate session id upon successful login) _____ / 1%
   - Validate the authentication token before revealing and executing admin features _____ / 3%
     - Create an Express middleware that executes before all protected `/admin` routes
     - Given proper validations, allow access to admin panel and executions of admin features
     - Otherwise (e.g. incorrect token), redirect back to the *login page*
     - Protected routes refer to all `/admin/*` and `/admin/api/*` except those `/admin/login`

- Provide a logout feature that void the authentication token _____ / 1%

6. Branch out **phase4a** in your repository, where TAs can checkout for inspection

PHASE 4B: NETWORK SECURITY (DEADLINE: MAR 30, 2015, 5PM) (SUBTOTAL: 5%)

In this phase, you will setup the end-to-end HTTPS support to defend against Man-in-the-middle attacks. Instructions and details can be found in the sample code repo and tutorial notes.

1. Apply SSL certificate for `store[01-80].ierg4210.org`
   - Certificate Application _____ / 2%
     - Apply a 90-day free certificate from FreeSSL.su (or another CA if you like)
     - TAs will help authorize your application only if you choose admin@ierg4210.org for domain control validation
     - Reminder: the application process can take more than a day, so apply early!!
     - Reminder: keep your secret key secure, or you will need to reapply a new certificate
   - Certificate Installation
     - Install the issued certificate at Elastic Beanstalk load balancer _____ / 1%
     - Enforce the admin panel /admin is always served at, and redirected as needed: _____ / 2%
       https://store[01-80].ierg4210.org/admin
       (Given your github repo is shopXX, TAs will map storeXX.ierg4210 to your application URL)

PHASE 5: SECURE CHECKOUT FLOW (DEADLINE: APRIL 12, 2015, 5PM) (SUBTOTAL: 15%)

This is a challenging phase, yet the most critical one to escalate your website to a professional level. :)

1. Sign up at Paypal and https://developer.paypal.com/webapps/developer/applications/myapps _____ / 1%
   - You will be issued a client_id and client_secret for API use
   - You will have two new accounts, facilitator (merchant) and buyer automatically assigned based on your email; change passwords, use the buyer account to pay, and the other one to receive payment
2. Create a payment table (or a separate DB with only one payment table) _____ / 1%
   - Required columns: *payid (primary key), userid, paymentId, state, dateCreated*
   - Required index: (userid, paymentId), (state)
3. Add a *Checkout* button to your shopping list implemented in phase 3 _____ / 1%
   - Once clicked, submit the shopping list to /checkout (i.e., with item's pids and qtys)
4. *Checkout page* hosted at /checkout _____ / 4%
   - If there is no valid user session, save the shopping list tentatively, redirect to /account/login/checkout.
   - Create a payment token using paypal.payment.create()
   - Update DB with the userid together with the payment id given by paypal
   - Redirect the user to the paypal checkout page (approval_url given by paypal)
     - The user will finally get redirected back to either thank you or error page
5. *Error page* hosted at /checkout/error (used as cancel_url for paypal) _____ / 1%
   - Tell the user the payment is cancelled or has encountered an error
   - Give a [Retry] hyperlink to the payment URL again (hint: query string "token")
6. *Thank you page* hosted at /checkout/thankyou (used as return_url for paypal) _____ / 2%
   - Claim the payment by using paypal.payment.execute()
   - If state == approved, update DB and thank you the user. Otherwise, redirect to the error page
7. *User login page* extended from phase 4A and hosted at /account/login/:action(\w+) _____ / 2%
   - Upon successful login, redirect back to /checkout given :action is checkout; otherwise, /account
8. *User Account page* hosted at /account _____ / 3%
   - Display a table showing the user-specific paymentId and state in chronological order
   - When clicked a paymentId,
     - Retrieve the items purchased using paypal.payment.get()
     - Display the total price and the items purchased including the quantity, price, name, photo
9. Branch out **phase5** in your repository, where TAs can checkout for inspection

For steps 4-6, integrations with Paypal using the paypal-rest-sdk npm needed (check their documentations, and the sample code at https://gist.github.com/adon-at-work/48e77447fb0513ec375c)

PHASE 6: EXTENSIONS (DEADLINE: DUE BEFORE YOUR DEMO) (SUBTOTAL: 8%, BONUS: 8% MAX)

You can choose any combinations of the following items to implement for 8%. At most 8% will be awarded for bonus.

1. Support multi-level categories for products (should update both frontend and backend) _____ / 3%
2. Support pagination / AJAX infinite scroll when browsing products in the main page _____ / 3%
3. Using AJAX when browsing categories and products in the main page _____ / 4%
    - Use location.assign to reflect the state of current view
    - The corresponding view is shown upon accessing the new URL in a new tab
4. Apply Search Engine Optimized (or user-friendly) URLs on frontend _____ / 2%
    - Include the name of categories and products into the URLs:
      e.g. /[catid]-[catname]/ for browsing products under the category [catid]
      e.g. /[catid]-[catname]/[pid]-[pname]/ for browsing product details of [pid]
2. Support HTML5 Drag-and-drop image upload in the admin panel _____ / 2%
    - Create and Highlight a dropping zone when a file is being dragged to the zone
    - Display a thumbnail (i.e. smaller width and height) if the dropped file is an image; reject it otherwise
3. Support AJAX image upload in the backend portal _____ / 3%
4. Support automatic image resizing for product images _____ / 3%
    - When a large image is uploaded, the image is automatically resized to produce a thumbnail image
    - In the main page, display thumbnails. In the product description page, display the large image.
5. Support Change of Password _____ / 3%
    - Must validate the current password first
    - Take the new passwords twice
    - Log user out after the password is changed
6. Support secure password reset through email _____ / 6%
    - A page that asks for email address for password recovery
    - Only if the email corresponds to an existing user, an email will be generated
    - In the email, a password recovery hyperlink includes a one-time random nonce
    - Password recovery validates the nonce before password generation or allow password change
7. Support multi-session management (demo: gmail, dropbox) _____ / 6%
    - Show the simultaneous logged-in sessions in the admin panel
    - Each session should be identified by an IP and allows logging out other sessions
    - Hints: Use DB to save valid authentication tokens
8. Mashup: Support Secure Authentication with Google or Facebook accounts _____ / 6%
    - Google: http://code.google.com/apis/accounts/docs/OAuth2.html
    - Facebook: https://developers.facebook.com/docs/authentication/
9. Setup HTTP Strict Transport Security (HSTS) for your HTTPS pages _____ / 1%
10. Build a user sign-up page hosted at /account/create _____ / 4%
    - Sent an email to the newly created user with an account activation link.
    - Account is activated only upon email confirmation
11. Support discounts when purchasing multiple quantities of a product type _____ / 10%
    - Create a DB table called discounts that store conditions for applying discounts
        - Conditions could include: "buy 2 get 1" and "buy $10@2, $6@1"
        - Refer to parknshop.com for reference and details
    - Update the UI to reflect the discounted price whenever the quantity conditions are met
    - Security: Make proper validations throughout the checkout process

PHASE 7A: SELF HACKING (DEADLINE: APR 15, 2015, 5PM) (SUBTOTAL: 3%)

It is critical to defend against potential attacks before they turn into reality when the website is open to the public.

1. Practice with the use of any *automated* vulnerability scanner _____ / 3%
    - Use an automated vulnerability scanner (e.g., Nikto, Skipfish or others). Fix your vulnerabilities, if any.
    - Scan ONLY your own website to see what common errors you could have made
    - Note: Beware of exceeding the bandwidth quota so that you are charged

PHASE 7B: PEER HACKING (MAY 6 5PM TO MAY 9 5PM) (SUBTOTAL: 10%, BONUS: 4% MAX)

This is to a game for you to practically learn security reviews. The peer hacking period will be held from May 6 5pm to May 9 5pm. Students will perform blackbox security reviews for each other's website that is hosted at http://store[01-80].ierg4210.org. Optionally, one may further their study on conducting whitebox code analysis on each other's github repository (i.e., https://github.com/ierg4210/shopXX to be made public on May 6 at around 5pm). If you found a vulnerability, report it directly to its github issue tracking page (i.e., https://github.com/ierg4210/shopXX/issues). Each student is entitled a base score of 8/10, while the highest possible score is 14 including bonus.

1. Clean your source code to erase any personal identifiable information, passwords and credentials
2. Perform *a* ethical hacking for shops of your classmates ___/10%(14%)
    - For a vulnerability in Student V's shop being reported by Student R_1:
        - Student R_2 reports a duplicate vulnerability after its first disclosure R_2: -3
        - Student R_1 reports and clearly specifies the problems R_1: +2 V: -2
            - Each student can report at most 4 security-related Student R_1 **MUST** follow the following format
                - Title: Prefixed with the vulnerability name (e.g., XSS, CSRF)
                - Content: Answer the following questions
                    1. Detailed steps to reproduce? (e.g., visit /admin2 without login)
                    2. What did you see? (e.g., led to a page that can delete a product)
                    3. What did you expect to see? (e.g., admin actions require authentication)
                    4. What is the possible fix? (e.g., delete the sample/testing page)
                - Attach at least one screen capture(s) as a proof and to aid illustration
        - Student V proves that R_1's report is invalid R_1: -3 V: +3
            - To dispute an issue, student V must redeploy his/her source code from github and demonstrate it to TA during final demo
        - Student V fixed the issue before his/her demonstration slot (or by May 10 if an issue is reported after the demo) V: +1.5
            - V MUST close the issue after fixing it
            - Fixing all reported issues guarantee a minimum score of 5 V: ≥5
    - For a security-UNrelated issue in Student V's shop being reported by Student R:

      Please understand that some weaker students might skip doing a part or two, considering them as bugs/issues to report are meaningless to the aim of this phase. So, unless your discovery is a kind of system flaw, otherwise, do not report it.
        - The score distribution is similar to security ones, but the effect is only 25% (i.e., 0.5/issue).
        - Each issue MUST be prefixed by the numbering of assignment requirements
            - Example number: [P1-1] stands for requirement 1 in phase 1
        - Browser compatibility issues are excluded, since it is not an assignment requirement.
        - Each student can report at most 3 non-security related bugs
    - Availability check by the automated IERG4210-robot [R: +0.5] V: -1
        - http://ierg4210.github.io/web/stores.html will report availability of all stores during the peer hacking period
        - Websites detected not functioning/available will be deducted 1 mark every 3 hours.
        - Student R can report V if his/her site is shown as available yet inaccessible by ordinary browsers and connections; only the first reporting student R is eligible for 1 mark
    - Student R is found causing high volume of traffic on others' websites R: -5
        - Includes but not limited to launching automated scan, DoS, or DDoS
        - Please shutdown the service to avoid being charged

This game is expected to be quite exciting! :) This is an internal ethical hacking exercise, hence, students whose websites are exploited or found vulnerable should not take it as offensive but a good learning opportunities. You may also like to learn from each other by looking into the source code of your peers. Students should respect each other and be polite in

any circumstances. In case of dispute, TA will be the final judge on the marking. Drastic circumstances (e.g. bullying) can result in penalties when the instructor deems it appropriate.

FINAL Q&A (SUBTOTAL: -75%)

- Random Question 1: Code-related _____%
- Random Question 2: Conceptual or Code-related _____%

SID: _____ TOTAL: _____ / 110%

MARKER RESPONSIBLE: _____